# Instagram Fake Account Detection

## Analisi degli attributi rilevanti nel riconoscimento automatico di account fasulli sulla piattaforma Instagram

Report del progetto per il Corso di Intelligenza Artificiale


**Gruppo Python in my boot**

Lorenzo Balugani, 0001039239

Alessandra Cosenza, 0001052401

Alberto Paparella, 0001041724



Corso di Laurea Magistrale in Informatica

Università degli studi di Bologna

Anno Accademico 2022-23

# Indice

# 1 Introduzione

Instagram è uno dei social più utilizzati dagli utenti più giovani della rete: la sua comunicazione, basata esclusivamente sulla condivisione di foto e video se non per brevi descrizioni, rende l'interazione più veloce e accattivante per un pubblico alla ricerca di un intrattenimento poco impegnativo. Ogni persona è in grado di creare uno o più account sulla piattaforma, permettendo di possedere sia un account personale, sia uno o più account dedicati ad altro: il profilo della propria azienda, una pagina dedicata ad una propria passione, come la fotografia, il bodybuilding o la propria fede calcistica, etc. La possibilità di creare un qualsiasi numero di account e la normalità nel trovare profili dedicati ad attività ha tuttavia permesso la diffusione di molti account creati con il solo scopo di promuovere attività fasulle (ad esempio, la promessa di investimenti in criptovalute) o di spacciarsi per qualcun altro (persona o azienda), così da ottenere informazioni personali di altri utenti. Inoltre, questo rende possibile per un profilo business (un imprenditore, un'azienda o un influencer) di fingere di avere un grosso seguito, misurato in like, commenti e follower, guadagnandosi così la fiducia di altri potenziali clienti, in un metodo simile alle recensioni finte di ristoranti, alberghi o negozi online, fenomeno noto in letteratura col nome di *fake engagement* [1]. Da questo emerge inoltre come la presenza di numerosi account fittizi rappresenti un problema che coinvolge l'utenza a 360 gradi: in primo luogo gli utenti comuni, soprattutto quelli più ingenui, che rischiano di cadere in truffe o frodi, ma anche le aziende ed i content creator, in quanto un soggetto malevolo appartenente alla stessa categoria potrebbe usufruire del fake engagement per ottenere un vantaggio sleale nei confronti dei competitor.
Al fine di risolvere il problema, è quindi utile identificare un modo il più accurato possibile per distinguere questi account da quelli reali: ciò fornirebbe in prima istanza una prevenzione in più per l'utente comune, che sarebbe meno soggetto al rischio di cadere in truffe o frodi, mentre dall'altro lato fornirebbe un primo strumento alle aziende e ai creatori di contenuti per individuare e segnalare eventuali account fittizi sfruttati dai competitor per migliorare il proprio seguito apparente, anche in maniera automatica.
La soluzione proposta prevede quindi l'individuazione delle caratteristiche più influenti nella distinzione fra account reali e account fasulli, con la tesi che solo un sottoinsieme specifico degli attributi a disposizione abbia un vero impatto in questa scelta, mentre altri potrebbero al contrario rischiare di rendere la classificazione non sufficientemente generica (*overfitting*). Si è scelto, come vincolo aggiuntivo ragionevole, di considerare solo attributi trivialmente ottenibili dagli utenti, ad esempio utilizzando la API resa disponibile da Instagram, o combinazioni di questi (facilmente ottenibili anche in maniera automatica). Per l'individuazione di questi attributi e la dimostrazione della loro validità è stata formalizzata una serie di esperimenti nei quali vari modelli di machine learning sono stati allenati sugli stessi dati (i.e., sugli stessi utenti), estratti a ogni iterazione casualmente dal dataset, ma considerando attributi differenti, dimostrando che la configurazione proposta ottiene in media dei valori migliori per quanto riguarda le metriche considerate.
La prima fase di progettazione ha visto una accurata revisione della letteratura esistente, che al momento della scrittura è tuttavia molto concisa, utilizzando come fonti arXiv, Google Scholar, Papers with code e Kaggle. Il primo contributo formale all'analisi del problema è fornito da [2], il quale fornisce un dataset per il riconoscimento di account fittizi e uno per gli account automatizzati e propone attributi derivati per la classificazione di questi, discernendo fra i due problemi e studiando il problema come due problemi di classificazione binari separati. Questi dataset sono tuttavia incompatibili, fornendo

attributi differenti per gli utenti. Date le descrizioni di account *Fake* e *Automated* fornite (queste tendono a differire in letteratura), abbiamo trovato come quest'ultima, riportata di seguito, sia quella che meglio si sposa con il problema in esame: si tratta di account che svolgono attività automatizzate come seguire, mettere like e commentare con hashtag specifici, o che contribuiscano al fenomeno del fake engagement, mostrando spesso comportamenti non organici (umani, naturali). Altro contributo di natura simile è fornito da [3], anch'esso legato al proprio dataset. A differenza del precedente, esso studia il problema come un problema di classificazione non binaria, dividendo gli utenti in 4 possibili classi: *active fake user*, *inactive fake user*, *spammer* e utenti reali. Tuttavia, a differenza del precedente, i dati sono forniti come un unico dataset, rendendo triviale la formalizzazione di un problema di classificazione binaria. Infine, altri contributi più recenti spostano il focus sul Deep Learning e sull'utilizzo di modelli specifici a soluzione del problema, ad esempio proponendo l'utilizzo di Generative Adversarial Network [4] oppure facendo affidamento alle recenti tecniche di Natural Language Processing [5]. Tuttavia, ciò esula dall'obiettivo di questo progetto, che pone l'accento sullo studio degli attributi che più influiscono alla scelta piuttosto che sull'utilizzo di una tecnica specifica, seppur più efficiente, in particolare se suddetta tecnica non è interpretabile.
Durante lo sviluppo del progetto, al fine di coordinare con efficacia il lavoro di gruppo, sono state effettuate riunioni con cadenza circa settimanale in modo da condividere i progressi individuali e discutere sia sui risultati ottenuti, sia su come affrontare la fase successiva. Per la stesura del report, scritto in LaTeX, è stato utilizzato lo strumento collaborativo Overleaf, mentre per quanto riguarda la condivisione del codice si è sfruttato come da richiesta lo strumento collaborativo GitLab. Le chiamate sono avvenute utilizzando la piattaforma Discord. Il lavoro è stato suddiviso fra i vari membri del gruppo come segue: Lorenzo si è occupato della gestione dei dati e della creazione dei dataset, nonchè degli esperimenti sulle reti neurali e relativo studio ablativo, Alessandra si è occupata della visualizzazione e dell'analisi dei dati, mentre Alberto si è occupato della parametrizzazione dei metodi di machine learning e relativo studio di comparazione. Ci teniamo tuttavia a specificare come questa suddivisione sia molto generica, e come ogni membro del gruppo abbia lavorato ad ogni parte e fase del progetto.
Con riferimento agli articoli sopracitati, l'indagine ha portato alla definizione di due insiemi di attributi con due diversi obiettivi: da una parte, trovare una combinazione di attributi per il dataset legato a [2] in grado di migliorare a livello sperimentale l'efficienza dei modelli di machine learning proposti, facendo uso di attributi scartati dagli autori o di combinazioni di quelli presenti, e dall'altra dimostrare che una riduzione degli attributi usati in [3] non comporti un calo significativo nelle performance degli stessi modelli. Questo studio ha quindi permesso di fornire ulteriori linee guida su quali attributi prendere in considerazione nella costruzione di un riconoscitore automatico di account fasulli sulla piattaforma, nonchè quali caratteristiche possano considerarsi come red flags per l'utente in casi sospetti. Tali linee guida sono inoltre brevemente sperimentate tentando una combinazione dei due dataset, con l'obiettivo di osservare la generalizzazione delle stesse su dataset con vincoli più stretti o dataset differenti.

# 2 Metodo proposto

## 2.1 Ricerca della soluzione

Al fine di indagare quali attributi svolgessero un ruolo rilevante nella discriminazione fra account reali e fasulli si è scelto di affidarsi ai dataset forniti da [3] e [2]. Il primo, da qui in seguito nominato `IJECE`, raccoglie informazioni su ben 65327 utenti, mentre il secondo, nominato `InstaFake`, soltanto 1400. Sebbene i due dataset siano di per sé simili, essi presentano delle differenze significative, e per questo si è scelto di affrontare il problema in maniera distinta e solo successivamente si è tentato di combinare i due dataset, in due modalità differenti descritte in seguito, al costo però di rinunciare ad alcuni attributi talvolta significativi. Lo scopo è di utilizzare i nostri metodi di studio (che saranno approfonditi in seguito in questa sezione) per confermare o meno i risultati dei relativi articoli, dopodiché usare gli stessi metodi su un dataset personalizzato (ossia con una scelta di attributi differente) per confrontare i due approcci ai dati, ed infine testare una possibile configurazione di attributi a partire da queste scelte tale per cui i due dataset siano combinabili, alla ricerca di linee guida ulteriori sulle caratteristiche più rilevanti per l'allenamento di modelli supervisionati a soluzione del problema.

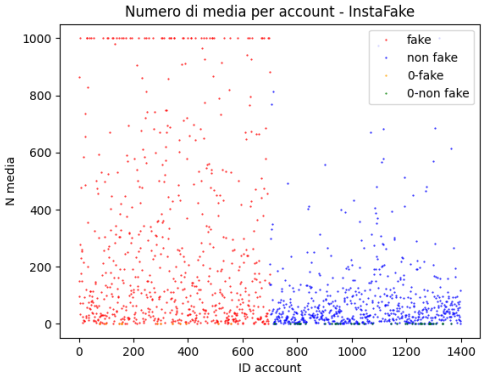
### 2.1.1 Analisi dei dati

I parametri che abbiamo ritenuto più rilevanti fra quelli presenti in entrambi i dataset sono i seguenti:

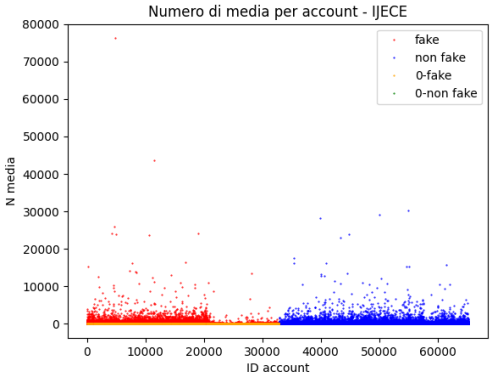
- Numero di media;
- Lunghezza della bio;
- Engagement rate per i like;
- Intervallo di tempo medio tra un post e il successivo;
- Numero di account seguiti dal profilo (*following*);
- Numero di account che seguono il profilo (*follower*).

Riportiamo quindi una rappresentazione della distribuzione dei dati relativamente a questi attributi in entrambi i dataset. Si è scelto di utilizzare un colore differente per rappresentare gli utenti, fittizi o meno, in cui il parametro di riferimento assume valore 0 (ad esempio, quando non sono presenti media o quando manca la bio).

Numero di media La distribuzione del numero di media per account è molto diversa per i due dataset: facendo riferimento alla Figura 1a risulta chiaro come il numero di media sia solitamente ben più alto negli account fake piuttosto che in quelli reali, con un'insolita tendenza nell'averne esattamente 1000 (cosa che potrebbe tuttavia essere dovuta ad un limite superiore posto nella creazione del dataset, di cui non siamo a conoscenza). Nel dataset IJECE invece la distribuzione sembra essere omogenea, con circa 10.000 account con un numero sotto la media e con nessun account reale con 0 foto.

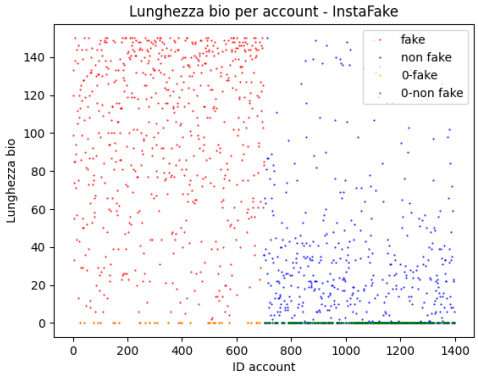



(a) Numero di media - InstaFake

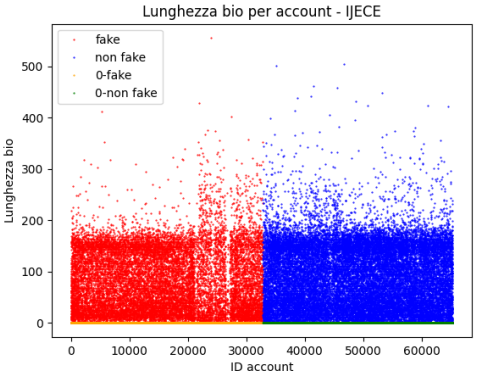



(b) Numero di media - IJECE

Figura 1: Numero di media

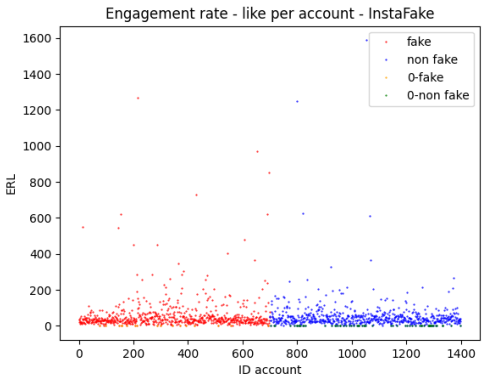



(a) Lunghezza della bio - InstaFake

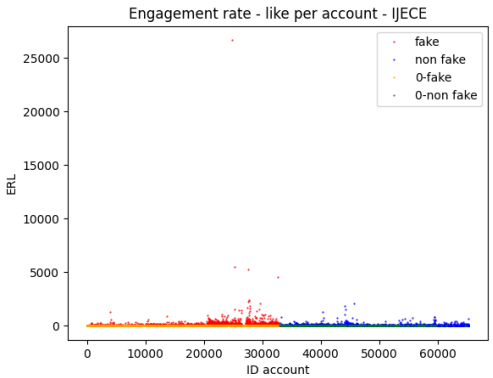



(b) Lunghezza della bio - IJECE

Figura 2: Lunghezza della bio

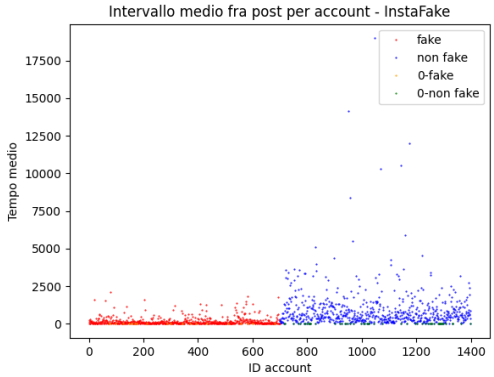



(a) ERL - InstaFake

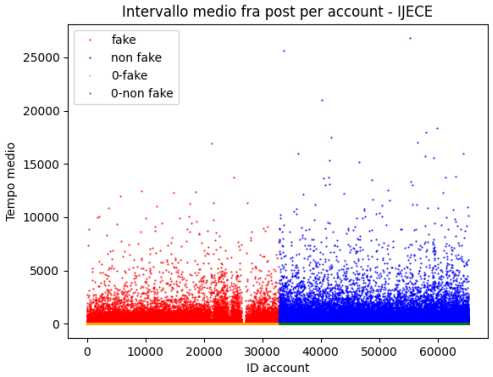



(b) ERL - IJECE

Figura 3: Engagement rate per i like

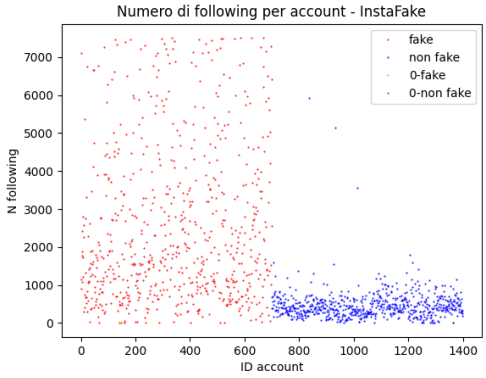



(a) Tempo medio - InstaFake

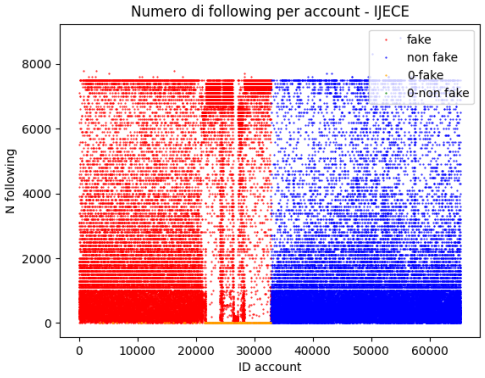



(b) Tempo medio - IJECE

Figura 4: Intervallo di tempo medio tra un post e il successivo

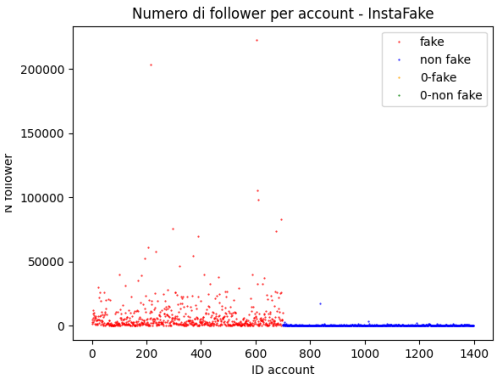



(a) following - InstaFake

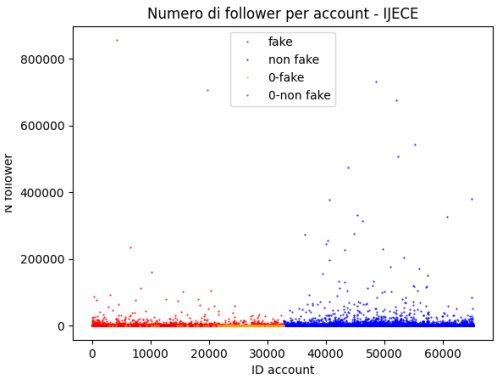



(b) following - IJECE

Figura 5: Numero di account seguiti dal profilo (*following*)




(a) follower - InstaFake




(b) follower - IJECE

Figura 6: Numero di account che seguono il profilo (*follower*)

Lunghezza della bio La lunghezza della bio è un parametro diverso da quanto previsto: infatti, Instagram prevede una lunghezza massima di 150 caratteri, come anche visibile dal dataset di InstaFake (Figura 2a), tuttavia il dataset IJECE (Figura 2b) prevede numerose bio oltre i 200 caratteri. Non siamo riusciti a trovare variazioni di lunghezza della bio nel corso degli anni, soltanto la possibilità di allungarla sfruttando il campo *indirizzo dell'azienda* che non prevede un numero massimo di caratteri. In ogni caso, nel dataset InstaFake una bio lunga è associata agli account falsi, con molti account veri senza bio, mentre nel dataset IJECE la media di lunghezza sembra omogenea.

Engagement rate per i like La media di like per post non sembra fornire informazioni molto rilevanti, se non che nel dataset IJECE sembra che siano molti di più gli account fasulli con 0 like rispetto a quelli reali (Figura 3).

Intervallo di tempo medio tra due post L'intervallo di tempo medio tra un post e il successivo sembra essere generalmente più alto negli account reali piuttosto che in quelli falsi (Figura 4).

Following Il numero di account seguiti da account falsi sembra essere molto più alto nel dataset InstaFake (Figura 5a), così come nel dataset IJECE (Figura 5b), anche se in maniera meno marcata.

Follower La rappresentazione in Figura 6 è una riduzione di quella totale in quanto vi erano rispettivamente 1 e 4 account non fake oltre il milione di follower (presumibilmente dei vip con account verificati) che disturbavano la visualizzazione di tutti gli altri account. Ora è ben visibile come il numero di follower nel primo dataset siano ben più alti per gli account falsi; viceversa, anche se in modo meno marcato, per IJECE.

Conclusioni L'analisi dei dati mostra come InstaFake presenti differenze marcate tra dati appartenenti a classi differenti, il che lo rende più *comprensibile* per un algoritmo di Machine Learning rispetto a quanto si vede con IJECE.

### 2.1.2 Algoritmi utilizzati

Per gli esperimenti sono stati presi in considerazione i seguenti metodi:

- Decision Tree;
- Random Forest;
- Support Vector Machine;
- Naive Bayes;
- Logistic Regression;
- Multilayer Perceptron.

Vediamo nel dettaglio questi metodi, le loro caratteristiche e come ne abbiamo fatto uso.

Decision Tree Un Decision Tree [6], o albero decisionale, è un modello di apprendimento automatico supervisionato utilizzato per compiti di classificazione e regressione, prendendo decisioni basate su una serie di condizioni. L'aspetto principale di un Decision Tree è la sua struttura ad albero, che consiste in nodi e archi. Ogni nodo rappresenta:

- una decisione o un test su un attributo specifico dei dati nel caso dei nodi interni, chiamati nodi di decisione;
- una etichetta di classe se si è nel caso della classificazione oppure i valori di outout nel caso dei nodi foglia.

Gli archi connettono i nodi e rappresentano i risultati dei test effettuati. Il processo di costruzione di un Decision Tree coinvolge principalmente due fasi: la selezione dell'attributo migliore per il test e la creazione dei sottoalberi corrispondenti ai risultati del test. Un criterio comune per la selezione dell'attributo migliore è l'indice di impurità, che misura quanto bene un attributo separa le diverse classi o valori target nei dati di addestramento. Alcuni degli indici di impurità comunemente utilizzati includono l'*indice di Gini* e l'*entropia*. Nella fase di addestramento, l'albero viene costruito iterativamente selezionando ricorsivamente gli attributi migliori e dividendo i dati in base ai test effettuati sugli attributi. Questo processo continua fino a quando non si verifica una delle seguenti condizioni: o tutti i dati in un sottoalbero appartengono alla stessa classe (per i problemi di classificazione) o i valori di output sono sufficientemente simili (per i problemi di regressione), oppure viene raggiunta una profondità massima predefinita dell'albero, oppure il numero di campioni in un nodo è inferiore a una soglia predefinita. I Decision Tree offrono diversi vantaggi: infatti, la struttura ad albero è intuitiva e può essere facilmente interpretata dagli umani, rendendo i risultati spiegabili; possono inoltre essere utilizzati con dati sia numerici che categorici senza richiedere molte pre-elaborazioni, nonchè catturare interazioni complesse tra variabili senza richiedere l'ingresso manuale di interazioni esplicite. Tuttavia, i Decision Tree possono anche essere soggetti al problema dell'*overfitting*, specialmente quando l'albero diventa molto profondo e si adatta troppo ai dati di addestramento. Per mitigare questo problema, spesso si utilizzano tecniche di potatura (*pruning*) che semplificano l'albero e ne migliorano la generalizzazione sui dati di test. Per gli esperimenti è stata utilizzata la classe `sklearn.tree.DecisionTreeClassifier` offerta dal pacchetto *sklearn*, il quale fa uso di una versione ottimizzata dell'algoritmo di `CART` [7], la quale al momento non supporta variabili categoriche.

Random Forest Le Random Forest [8], o foreste casuali, sono un'evoluzione dei Decision Tree utilizzate anch'esse per problemi di classificazione e regressione, offrendo maggiore accuratezza e resistenza all'overfitting rispetto ai singoli Decision Tree. Una Random Forest è costituita da un insieme (o *forest*) di Decision Tree, ognuno dei quali viene addestrato su un sottoinsieme casuale dei dati di addestramento selezionato tramite campionamento con sostituzione, noto anche come *bootstrapping*. Inoltre, durante la costruzione di ciascun albero, a ogni divisione del nodo, viene considerato solo un sottoinsieme casuale degli attributi anziché tutti gli attributi disponibili. Questo processo di campionamento casuale e di limitazione degli attributi durante la costruzione degli alberi è ciò che conferisce alle Random Forest la loro capacità di generare modelli potenti e diversificati. Infatti, poiché ciascun albero è addestrato su un sottoinsieme casuale dei dati, il problema dell'overfitting è ridotto, mentre l'aggregazione degli output di molti alberi riduce la varianza complessiva del modello, migliorando così la capacità

di generalizzazione su dati non visti. A causa della natura aggregata delle previsioni, le Random Forest sono meno sensibili agli outlier e ai dati rumorosi rispetto ai singoli Decision Tree. Inoltre, poiché questi possono essere addestrati indipendentemente, le Random Forest possono beneficiare di un'efficace parallellizzazione, accelerando il processo di addestramento. Tuttavia, la loro efficacia dipende anche dalla scelta di parametri come il numero di alberi e la profondità massima degli alberi. Per gli esperimenti è stata utilizzata la classe `sklearn.ensemble.RandomForestClassifier` fornita dal pacchetto *sklearn*, specificando una profondità massima (*max_depth*) di 2.

Support Vector Machine Le Support Vector Machine (SVM) [9] sono uno strumento di apprendimento automatico supervisionato che mira a trovare il miglior iperpiano o superficie di decisione che separa i dati in diverse classi. In una classificazione binaria, come nel nostro caso, un iperpiano è un piano di dimensione $(n-1)$, dove $n$ è il numero di caratteristiche dei dati. L'obiettivo è trovare l'iperpiano che massimizza il margine tra le classi, ovvero la distanza tra l'iperpiano di separazione e i vettori di supporto delle classi, definiti come le osservazioni che si trovano più vicine all'iperpiano di separazione. Un margine più ampio suggerisce una maggiore generalizzazione su nuovi dati, e riduce il rischio di errore di classificazione sui dati futuri. L'addestramento di un modello SVM coinvolge l'ottimizzazione di parametri come il parametro di regolarizzazione $C$ e il tipo di *kernel*, una funzione per mappare i dati in uno spazio delle caratteristiche più elevato in cui possano essere separati linearmente per affrontare problemi di classificazione più complessi (in molti casi, i dati non sono linearmente separabili nell'ambiente originale), la cui scelta appropriata può influenzare le prestazioni del modello. Lato negativo di questo strumento è infatti l'accurata messa a punto dei parametri, oltre ad essere computazionalmente impegnativo per grandi quantità di dati. Le SVM possono essere estese per affrontare problemi di classificazione multi-classe utilizzando approcci come One-vs-All (OvA) o One-vs-One (OvO), in cui vengono addestrati diversi classificatori binari per ciascuna coppia di classi, e possono essere utilizzate anche per problemi di regressione, dove l'obiettivo è prevedere un valore numerico invece di una classe. In questo caso, l'SVM cerca di adattare una funzione che abbia un margine ampio rispetto ai punti di dati più vicini. Per gli esperimenti è stata utilizzata la classe `sklearn.svm.LinearSVC` (Linear Support Vector Classification) fornita da sci-kit learn, la cui implementazione è basata su `liblinear` [10], a differenza della versione classica (`sklearn.svm.SVC`, C-Support Vector Classification) basata invece su `libsvm` [11], permettendo una miglior scalabilità con un numero maggiore di campioni (il tempo di allenamento per la versione classica scala infatti quadraticamente col numero di campioni). Essa fa uso di un kernel lineare. È stato inoltre specificato di risolvere il problema di ottimizzazione originale e non quello duale, avendo a disposizione un numero maggiore di campioni che di caratteristiche, e una tolleranza di $1e^{-5}$, mentre il parametro di regolarizzazione è stato mantenuto a 1.

Naive Bayes Naive Bayes è un algoritmo di classificazione basato sul teorema di Bayes, il quale afferma che la probabilità di un'ipotesi data l'evidenza può essere calcolata utilizzando la probabilità dell'evidenza data l'ipotesi, la probabilità dell'ipotesi e la probabilità dell'evidenza. È chiamato *naive* (ingenuo) poiché assume che tutte le variabili di input siano indipendenti tra loro, semplificando il calcolo delle probabilità condizionali, poiché la probabilità congiunta delle caratteristiche può essere approssimata come il prodotto delle probabilità delle caratteristiche individuali. Nel contesto della classificazione, l'algoritmo Naive Bayes calcola la probabilità di appartenenza di un'istanza a una determinata

classe data l'osservazione delle sue caratteristiche secondo la seguente formula:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Dove $P(C|X)$ è la probabilità che l'istanza appartenga alla classe $C$ dato l'insieme di caratteristiche $X$, $P(X|C)$ è la probabilità di osservare l'insieme di caratteristiche $X$ dato che l'istanza appartiene alla classe $C$, $P(C)$ è la probabilità a priori della classe $C$ e $P(X)$ è la probabilità dell'insieme di caratteristiche $X$. Esistono diverse varianti di Naive Bayes basate su diverse assunzioni riguardanti la distribuzione di probabilità $P(x_i|C)$. Per esempio, essa può seguire una distribuzione di Bernoulli:

$$P(x_i|C) = P(x_i = 1|y)x_i + (1 - P(x_i = 1|y))(1 - x_i)$$

Penalizzando esplicitamente la non-occorrenza di una caratteristica $i$ che sia un indicatore di una classe $C$, oppure una distribuzione Gaussiana:

$$P(x_i|C) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

Per gli esperimenti sono sono state utilizzate le classi `sklearn.naive_bayes.BernoulliNB` e `sklearn.naive_bayes.GaussianNB` fornite da sci-kit learn, le quali implementano rispettivamente le varianti di Naive Bayes sopracitate.

Logistic Regression Il metodo di Logistic Regression [12], o regressione logistica, è un algoritmo di apprendimento supervisionato utilizzato per la classificazione binaria. Esso fa uso della funzione logistica (o sigmoide) per mappare l'input a un valore compreso tra 0 e 1, che rappresenta la probabilità dell'evento di classe positiva. Questa funzione è definita come:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Dove $x$ è una combinazione lineare delle variabili di input ponderate dai coefficienti del modello. Durante la fase di addestramento, il Logistic Regressor utilizza il metodo della massima verosimiglianza per stimare i coefficienti che massimizzano la probabilità di osservare i dati di addestramento. Questo viene fatto minimizzando la funzione di perdita, solitamente una log-loss. Una volta addestrato, il Logistic Regressor può essere utilizzato per effettuare previsioni su nuovi dati di input. L'output prodotto è una probabilità predetta che può essere mappata in una classe binaria utilizzando una soglia di decisione. Ancora una volta ci siamo affidati alla libreria *sklearn* utilizzando la configurazione di default, che fa uso dell'algoritmo `lbfgs` [13] (Large-scale Bound-constrained Optimization), utilizzando al massimo 5000 iterazioni.

Multilayer Perceptron Un Multilayer Perceptron (MLP) [14] è una rete neurale feed-forward composta da diversi layer di neuroni artificiali. Gli MLP possono venire definiti come una sottocategoria delle reti neurali profonde (DLP). Una rete neurale è un insieme di neuroni artificiali, astrazioni del neurone organico, collegati tra di loro tramite un insieme di connessioni, ciascuna delle quali ha associato un determinato peso. L'apprendimento supervisionato di una rete neurale consiste nel dividere i dati a disposizione in due gruppi, uno di addestramento e uno di validazione, e in quanto supervisionato

questi dati sono *etichettati*, ovvero è noto l'output desiderato della rete. Nel corso dell'addestramento, i pesi dei collegamenti tra neuroni verranno fatti variare per tentare di minimizzare l'errore, ovvero la distanza tra l'output predetto e quello reale, utilizzando le informazioni presenti nel gruppo di addestramento. Successivamente, si bloccano i pesi e si verifica la presenza di overfitting (la rete ha interiorizzato caratteristiche presenti solo nel training set) oppure di underfitting (la rete non ha interiorizzato abbastanza le caratteristiche dei dati) usando la rete addestrata sull'insieme di validazione.
Nel nostro caso, si è scelto di utilizzare come libreria *Tensorflow* e le sue classi `Input, Dense, Model` che consentono la definizione di reti neurali profonde, il loro addestramento e la loro validazione. Una particolarità degli esperimenti svolti con gli MLP risiede nell'avere dataset *fissati*, e vengono fornite reti già addestrate, salvate nel formato `.h5`.
I vantaggi di un MLP sono numerosi, ad esempio dopo un lungo periodo di addestramento una tantum le predizioni possono essere eseguite molto rapidamente, ma ha anche diversi svantaggi, come l'essere opachi (non è possibile comprendere cosa abbia portato la rete ad un verdetto) e la qualità della rete è legata alla qualità dell'addestramento, con la possibilità di avviare la rete con una selezione di pesi iniziale *sfortunata*.

### 2.1.3 Nomenclatura dei dataset

Sono di seguito riportati i nomi e le caratteristiche dei dataset utilizzati all'interno di questo progetto e a cui faremo da qui in avanti:

- InstaFake-Paper: dataset di partenza, rappresenta i risultati riportati nell'articolo [2] e viene usato a scopo comparativo. Nello specifico, la definizione dell'articolo di *Automated Account* corrisponde con quella dell'articolo [3] di *Fake Account*, ed è quindi stato utilizzato il relativo dataset;
- IJECE-Paper: dataset di partenza, rappresenta i risultati riportati nell'articolo [3] e viene usato a scopo comparativo;
- InstaFake-Default: dataset su cui sono state eseguite le nostre misurazioni *default*, con le stesse caratteristiche indicate nell'articolo [2];
- IJECE-Default: dataset su cui sono state eseguite le nostre misurazioni *default*, con le stesse caratteristiche indicate nell'articolo [3];
- InstaFake-Custom: dataset personalizzato basato su InstaFake-Default;
- IJECE-Custom: dataset personalizzato basato su IJECE-Default;
- InstaFake-Compatibility: dataset personalizzato basato su InstaFake-Custom, costruito usando feature comuni con IJECE-Custom;
- IJECE-Compatibility: dataset personalizzato basato su IJECE-Custom, costruito usando feature comuni con InstaFake-Custom;
- Combo-partial: dataset costituito da 2.800 utenti provenienti da IJECE-Compatibility e InstaFake-Compatibility in ugual misura;
- Combo-full: dataset contenente tutti gli utenti contenuti in IJECE-Compatibility e InstaFake-Compatibility.

### 2.1.4 Contenuto dei dataset personalizzati

InstaFake-Custom Per ottenere il dataset sono stati aggiunti attributi presenti nel dataset ma assenti all'interno dell'articolo [2]. Sono stati inoltre derivati altri valori da quelli a disposizione.

- nmedia: il numero di post prodotti dall'utente;
- biol: la lunghezza del campo *bio* dell'utente;
- url: presenza/assenza di url all'interno del campo *bio*;
- nfollowing: numero di utenti seguiti dall'utente;
- nfollower: numero di utenti che seguono l'utente;
- erl: rapporto tra numero di like, numero di media e numero di follower;
- erc: rapporto tra numero di commenti, numero di media e numero follower;
- avgtime: tempo medio (in ore) tra un post e il successivo;
- mediaLikeNumbers: numero medio di like per post;
- mediaHashtagNumbers: numero medio di hashtag per post;
- followerToFollowing: rapporto tra utenti che seguono e utenti seguiti;
- mediaCommentNumbers: numero medio di commenti per post;
- mediaCommentsAreDisabled: rapporto tra nmedia e il numero di post per cui i commenti sono disattivati;
- mediaHasLocationInfo: rapporto tra nmedia e il numero di post che contengono informazioni sulla posizione;
- hasMedia: presenza/assenza di post;
- userHasHighlightReels: presenza/assenza di reels nel profilo dell'utente.
- userTagsCount: numero di tags usati dall'utente;
- usernameLenght: lunghezza dello *username*;
- usernameDigitCount: numero di cifre presenti nello *username*;
- fake: tipologia di utente.

IJECE-Custom Per ottenere il dataset sono stati rimossi attributi considerati *dannosi*, o comunque irrilevanti, per le performance in seguito ad uno studio condotto durante le prime fasi del progetto partendo dal dataset dell'articolo [3]. Sono stati inoltre aggiunti valori derivati da quelli a disposizione.

- nmedia: il numero di post prodotti dall'utente;
- nfollower: numero di utenti che seguono l'utente;
- nfollowing: numero di utenti seguiti dall'utente;
- biol: la lunghezza del campo *bio* dell'utente;
- pic: presenza/assenza di un immagine di profilo;
- url: presenza/assenza di link esterni nel campo *bio*;
- cl: lunghezza media della descrizione di un post;
- cz: percentuale di descrizioni di dimensione prossima a 0;
- erl: rapporto tra numero di like, numero di media e numero di follower;
- erc: rapporto tra numero di commenti, numero di media e numero di follower;
- pr: uso medio di parole *promozionali* negli hashtag (ad esempio, *giveaway*, *quiz*);
- fo: uso medio di parole come *follow, like, f4f*;
- cs: similarità cosenica tra tutte le coppie di post;
- avgtime: tempo medio (in ore) tra un post e il successivo;
- followerToFollowing: rapporto tra utenti che seguono e utenti seguiti;
- hasmedia: presenza/assenza di media;
- fake: tipologia di utente.

Dataset Compatibility e Combined Per ottenere i dataset sono state selezionate le caratteristiche comuni tra InstaFake-Custom e IJECE-Custom.

- nmedia: il numero di post prodotti dall'utente.
- biol: la lunghezza del campo *bio* dell'utente.
- url: presenza/assenza di url all'interno del campo *bio*.
- nfollowing: numero di utenti seguiti dall'utente.
- nfollower: numero di utenti che seguono l'utente.
- erl: rapporto tra numero di like, numero di media e numero di follower.
- erc: rapporto tra numero di commenti, numero di media e numero di follower.

- avgtime: tempo medio (in ore) tra un post e il successivo.
- mediaHashtagNumbers: numero medio di hashtag per post.
- followerToFollowing: rapporto tra utenti che seguono e utenti seguiti.
- hasMedia: presenza/assenza di post.
- fake: tipologia di utente.

## 2.2 Descrizione del metodo per la misurazione delle performance

Per misurare le performance sono state utilizzate le seguenti metriche:

- **accuracy**, ovvero il rapporto fra il numero di predizioni corrette restituite dal modello sul totale
$$\frac{TP + TN}{TP + FP + TN + FN}$$
- **precision**, ovvero la proporzione di classificazioni positive corrette
$$\frac{TP}{TP + FP}$$
- **recall**, ovvero la proporizione di valori positivi identificati correttamente
$$\frac{TP}{TP + FN}$$
- **F1-score**, ovvero la media armonica fra precision e recall
$$2\frac{precision \cdot recall}{precision + recall}$$

Dove **TP (True Positives)** rappresenta il numero di predizioni positive corrette, **FP (False Positives)** rappresenta il numero di predizioni positive errate, **TN (True Negatives)** rappresenta il numero di predizioni negative corrette e **FN (False Negatives)** rappresenta il numero di predizioni negative errate.
Sono stati inoltre realizzati esperimenti per testare la variazione di tali statistiche al variare dei dati utilizzati, secondo il seguente algoritmo, che vale per tutti gli esperimenti eccezion fatta per quello sui percettroni multistrato:

1. Vengono importati i due dataset contenenti rispettivamente gli utenti **reali** e gli utenti **fittizi**, conservando gli attributi originali presentati nel rispettivo articolo;
2. Viene effettuato lo *shuffle* di entrambi i dataset e vengono estratti i dataset di allenamento e di test, estraendo rispettivamente i primi `PERCENT_TRAIN * len(dataset)` e i rimanenti `(1 - PERCENT_TRAIN) * len(dataset)` da entrambi, unendoli ed effettuando un ulteriore *shuffle*;
3. Vengono creati due ulteriori dataset contenenti gli stessi utenti presenti nel dataset di allenamento e di test, ma eliminando (e/o aggiungendo) alcuni attributi;

4. Viene allenato il rispettivo modello (ad esempio, un albero decisionale) sul dataset di allenamento originale e valutato sul dataset di test originale;

5. Viene allenato il rispettivo modello sul nuovo dataset di allenamento e valutato sul nuovo dataset di test;

6. Vengono conservati i valori delle metriche di valutazione e l'esperimento è ripetuto dal punto 2 per un numero consono di iterazioni;

7. Infine, viene restituita una media delle metriche di valutazione sia per i modelli realizzati a partire dai dataset originali, sia per quelli realizzati a partire dai dataset modificati, così da provare l'impatto della modifica degli attributi da noi proposto.

Scopo dell'esperimento è dimostrare come lo stesso modello allenato con gli stessi utenti ma considerando gli attributi proposti invece che quelli originali sia in media più efficiente rispetto a quello di partenza, e che quindi la configurazione proposta è migliore rispetto allo stato dell'arte.
Per quanto riguarda gli esperimenti sui percettroni multistrato, vengono forniti per ogni dataset un training set e un test set, insieme alle reti che durante il nostro studio hanno performato meglio. Training set e test set vengono ottenuti in modo analogo a quanto indicato nei punti 1 e 2 dell'algoritmo precedentemente descritto, e in seguito salvati per poi venire utilizzati in tutti gli esperimenti MLP successivi. Dopo aver caricato il modello, viene eseguito il test di valutazione e i risultati vengono restituiti come negli altri esperimenti.

# 3 Risultati sperimentali

## 3.1 Istruzioni per la dimostrazione

Questa sezione contiene le istruzioni per configurare ed eseguire il progetto. Queste istruzioni sono state pensate per un sistema Unix-based, di conseguenza se si vuole installare il progetto su un sistema operativo Windows-based è necessario sostituire i forward-slash (`/`) con dei backwards-slash (\) nei percorsi, e usare la cartella `Scripts/` al posto di `bin/` per eseguire il progetto tramite il `venv`. In base a com'è stato installato python, potrebbe essere necessario sostituire `python` nei comandi con `python3`, ed è necessario che il modulo `venv` sia installato. E' necessario utilizzare Python 3.

1. Eseguire il download del progetto dal `repository` GitLab oppure usare il comando `git clone https://gitlab.com/Evil_Balu/ai-pythoninmyboot`.

2. Estrarre il contenuto della cartella e aprire un terminale nella radice del progetto.

3. Creare un `venv` python con il comando `python -m venv venv`, che creerà un ambiente virtuale nella cartella del progetto.

4. Installare le dipendenze con il comando `./venv/bin/pip install -r ./requirements.txt`.

5. Eseguire il file `main.py` con `./venv/bin/python ./main.py`

Una volta avviato lo script `main.py`, è possibile scegliere quali esperimenti eseguire ed il numero di iterazioni per ciascun esperimento tramite un'interfaccia utente testuale.

## 3.2 Elenco tecnologie usate

Tutte le librerie utilizzate dal progetto sono contenute all'interno del file `requirements.txt`, con relativa versione, e sono riportate di seguito. Per installarle, seguire sottosezione 3.1.

- scikit-learn==1.2.2
- pip∼=21.2.3
- wheel∼=0.37.1
- py∼=1.11.0
- lxml∼=4.7.1
- pytz∼=2021.3
- pytest∼=6.2.5
- setuptools∼=57.4.0
- scipy∼=1.10.1
- joblib∼=1.2.0
- pandas∼=1.5.3
- MarkupSafe∼=2.0.1
- python-dateutil∼=2.8.2
- scikit-learn∼=1.2.2
- threadpoolctl∼=3.1.0
- six∼=1.16.0
- docutils∼=0.19
- tensorflow∼=2.12.0
- matplotlib
- keras∼=2.12.0
- numpy

## 3.3 Risultati della configurazione migliore

Sono di seguito riportate le configurazioni dei vari metodi con cui sono stati prodotti i risultati migliori, riportati nelle sezioni successive:

- **Decision Tree**: viene utilizzata la configurazione di default, ottenendo ottimi risultati senza necessità di fine-tuning.
- **Random Forest**: viene utilizzata una *max_depth* di 2, che già permette di ottenere ottimi risultati, per evitare tempi di allenamento troppo lunghi.
- **Support Vector Machine**: viene specificato di non considerare il problema di ottimizzazione duale in quanto abbiamo a disposizione un numero maggiore di istanze rispetto al numero di attributi, una tolleranza di $1e^{-5}$ e come parametro di regolarizzazione $C$ il valore di default a 1.0.
- **Naive Bayes (Bernoulli dist.)**: viene specificato un parametro di smoothing additivo di Laplace/Lidstone *alpha* di 500.0 e specificato di non variare questo valore ($force\_alpha = True$).
- **Naive Bayes (Gaussian dist.)**: viene utilizzata la configurazione di default in quanto non siamo stati in grado di trovare configurazioni migliori.
- **Logistic Regression**: viene specificato di non considerare il problema di ottimizzazione duale in quanto abbiamo a disposizione un numero maggiore di istanze rispetto al numero di attributi, una tolleranza di $1e^{-5}$ e come parametro di regolarizzazione $C$ il valore di default a 1.0; viene inoltre specificato un numero massimo di iterazioni ($max\_iter$) di 5000 in modo da assicurare la convergenza nei casi considerati.

- **Multilayer Perceptron**: 2 layer interni da 8 neuroni ciascuno con relu, learning rate 0.001, 100 epoche, batch size 16, riduzione learning rate basata su loss con fattore 0.5 dopo 4 epoche, funzione di loss "Binary Crossentropy".

Per maggiori informazioni sulla scelta di questi parametri rimandiamo allo studio di comparazione dei modelli nella sottosezione 3.5.

Media dei risultati ottenuti coi vari metodi per ogni dataset Dalla Tabella 1 si evince che il dataset che performa meglio sotto qualunque aspetto è quello di InstaFake con una selezione di attributi da noi ricercata. Questo è probabilmente dovuto sia agli attributi, sia al fatto di avere un dataset ristretto e che presenta delle differenze molto marcate tra i tipi di account (come evidenziato anche dalla visualizzazione dei dati in sottosottosezione 2.1.1). Va inoltre sottolineato che il paper di IJECE presenta dei risultati migliori rispetto alle nostre esecuzioni, tuttavia non viene indicato come replicare tali risultati, perciò vengono riportati solo a scopo informativo anziché comparativo. Infine, l'osservazione più importante per la nostra tesi, è che le configurazioni custom superano quelle default sia per Accuracy che per F1-Score per entrambi i dataset.

| | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| *InstaFake (paper)* | | *0.792* | *0.804* | *0.784* |
| InstaFake (our default) | 0.8 | 0.824 | 0.896 | 0.835 |
| InstaFake (custom) | **0.872** | **0.868** | 0.901 | **0.881** |
| InstaFake (compatibile) | 0.864 | 0.862 | **0.908** | 0.875 |
| *IJECE (paper)* | *0.828* | *0.836* | *0.828* | *0.827* |
| IJECE (our default) | 0.789 | **0.842** | 0.726 | 0.768 |
| IJECE (custom) | **0.793** | 0.829 | **0.746** | **0.782** |
| IJECE (compatibile) | 0.785 | 0.817 | 0.745 | 0.772 |
| Combinazione parziale | 0.733 | 0.764 | 0.662 | 0.684 |
| Combinazione totale | **0.772** | **0.821** | **0.701** | **0.747** |

Tabella 1: Media di tutte le metriche per i vari dataset

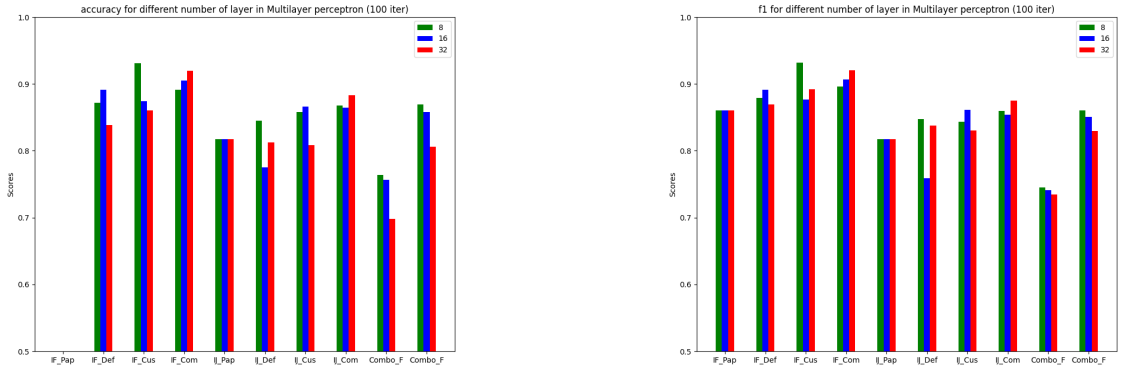
Media dei risultati ottenuti nei vari dataset per ogni metodo Dalla Tabella 2 si evince che i Decision Tree sembrano essere il metodo più appropriato per questo tipo di dati, con le Random Forest, le SVM e il MultiLayer Perceptron che restano comunque molto validi. Tuttavia, è invece evidente come i metodi di Naive Bayes non reggano mai il confronto con gli altri metodi.

| | Accuracy | Precision | Recall | F1 |
|---|---|---|---|---|
| Decision Tree | 0.876 | 0.875 | **0.879** | **0.876** |
| Random Forest | **0.878** | **0.909** | 0.844 | 0.874 |
| Support Vector Machine | 0.833 | 0.877 | 0.785 | 0.827 |
| Naive Bayes (Bernoulli) | 0.646 | 0.615 | 0.618 | 0.633 |
| Naive Bayes (Gaussian) | 0.693 | 0.706 | 0.782 | 0.703 |
| Linear Regression | 0.825 | 0.834 | 0.799 | 0.815 |
| Multilayer Perceptron | 0.811 | 0.888 | 0.796 | 0.854 |

Tabella 2: Media di tutte le metriche per i vari metodi

## 3.4 Studio di ablazione

Abbiamo tentato l'addestramento di reti con diversi numeri di neuroni al loro interno. Precisamente, abbiamo addestrato tre reti per ogni dataset a disposizione, ognuna con rispettivamente 8, 16 o 32 neuroni su ciascuno dei due livelli interni. I risultati sono riportati in Figura 7 e abbiamo deciso di mantenere la rete con i risultati migliori sui dataset che ci premevano di più, ossia quelli custom, quindi i risultati presentati nello studio di comparazione saranno quelli della rete con 8 neuroni per layer.




(a) Accuracy

(b) F1 score

Figura 7: Risultati per una rete MLP da 8, 16 e 32 neuroni per layer in 100 iterazioni

## 3.5 Studio di comparazione

Di seguito sono riportate graficamente le performance dei vari metodi sui diversi dataset, organizzate per metodo. Le performance sono state calcolate come una media fra 100 diverse esecuzioni, seguendo il procedimento descritto in sottosezione 2.2. In particolare, siamo interessati al confronto, per ogni metodo, fra le performance ottenute utilizzando i dataset con gli attributi originali (*IF_Def, IJ_Def*) e i dataset costruiti utilizzando le nostre proposte di attributi (*IF_Cus, IJ_Cus*), dando maggiore importanza ad accuracy, ovvero l'effettivo rapporto di predizioni corrette, e F1-score, in quanto media armonica fra precision e recall. Riportiamo quando possibile, a scopo puramente informativo, i risultati indicati negli articoli di riferimento; tuttavia, non è stato possibile replicare fedelmente gli esperimenti proposti, in quanto non sono specificati i parametri di configurazione dei classificatori. Riportiamo inoltre i risultati ottenuti utilizzando una configurazione di attributi compatibile per i due dataset (*IF_Com, IJ_Com*), ed i risultati ottenuti unendo i due dataset (*Combo_P*, utilizzando lo stesso numero di istanze provenienti da entrambi i dataset in modo da bilanciare il rispettivo contributo, e *Combo_F* utilizzando tutte le istanze provenienti da entrambi i dataset). Per una descrizione esatta di queste configurazioni riportiamo alla sezione 2.1.4. Infine, le Tab. 3, 4, 5, 6 riportano i risultati esatti di queste metriche per tutti i modelli usando entrambi i dataset, organizzati per metrica.

Decision Tree e Random Forest Figura 8 mostra i risultati ottenuti utilizzando i metodi di Decision Tree e Random Forest. Utilizzando i Decision Tree la configurazione custom performa meglio della configurazione di default in tutte le metriche con entrambi i dataset,

| | DT | RF | SVM | NB (B) | NB (G) | LR | MP |
|---|---|---|---|---|---|---|---|
| *InstaFake (paper)* | | | | | | | |
| InstaFake (our default) | 0.904 | 0.915 | 0.896 | 0.545 | 0.594 | 0.876 | 0.872 |
| InstaFake (custom) | 0.916 | **0.931** | **0.924** | **0.76** | 0.723 | **0.917** | **0.931** |
| InstaFake (compatibility) | **0.917** | 0.929 | 0.91 | 0.685 | **0.838** | 0.876 | 0.891 |
| *IJECE (paper)* | *0.883* | *0.901* | | | *0.731* | *0.809* | 0.817 |
| IJECE (our default) | 0.856 | 0.858 | 0.805 | 0.677 | 0.68 | **0.804** | 0.845 |
| IJECE (custom) | **0.857** | 0.859 | **0.793** | **0.679** | **0.706** | 0.799 | 0.858 |
| IJECE (compatibility) | 0.851 | **0.862** | 0.783 | 0.647 | 0.699 | 0.782 | **0.867** |
| Combo (partial) | 0.846 | 0.809 | **0.778** | 0.536 | 0.612 | **0.783** | 0.764 |
| Combo (full) | **0.85** | **0.842** | 0.774 | **0.642** | **0.653** | 0.777 | **0.869** |

Tabella 3: Accuracy per tutti i metodi con i vari dataset.

| | DT | RF | SVM | NB (B) | NB (G) | LR | MP |
|---|---|---|---|---|---|---|---|
| *InstaFake (paper)* | | | *0.91* | *0.85* | *0.51* | *0.8* | *0.89* |
| InstaFake (our default) | 0.903 | 0.942 | **0.96** | 0.526 | 0.619 | 0.9 | 0.834 |
| InstaFake (custom) | **0.918** | **0.947** | 0.944 | **0.735** | 0.696 | **0.929** | **0.929** |
| InstaFake (compatibility) | **0.918** | 0.943 | 0.933 | 0.629 | **0.832** | 0.858 | 0.857 |
| *IJECE (paper)* | *0.886* | *0.907* | | | *0.759* | *0.81* | *0.818* |
| IJECE (our default) | 0.853 | **0.945** | 0.822 | 0.746 | **0.807** | 0.812 | 0.842 |
| IJECE (custom) | **0.854** | 0.896 | **0.831** | **0.747** | 0.725 | **0.83** | **0.947** |
| IJECE (compatibility) | 0.848 | 0.907 | 0.812 | 0.75 | 0.698 | 0.789 | 0.921 |
| Combo (partial) | 0.846 | 0.819 | **0.866** | 0.718 | 0.664 | **0.818** | 0.808 |
| Combo (full) | **0.847** | **0.877** | 0.816 | **0.746** | **0.749** | 0.796 | **0.929** |

Tabella 4: Precision per tutti i metodi con i vari dataset.

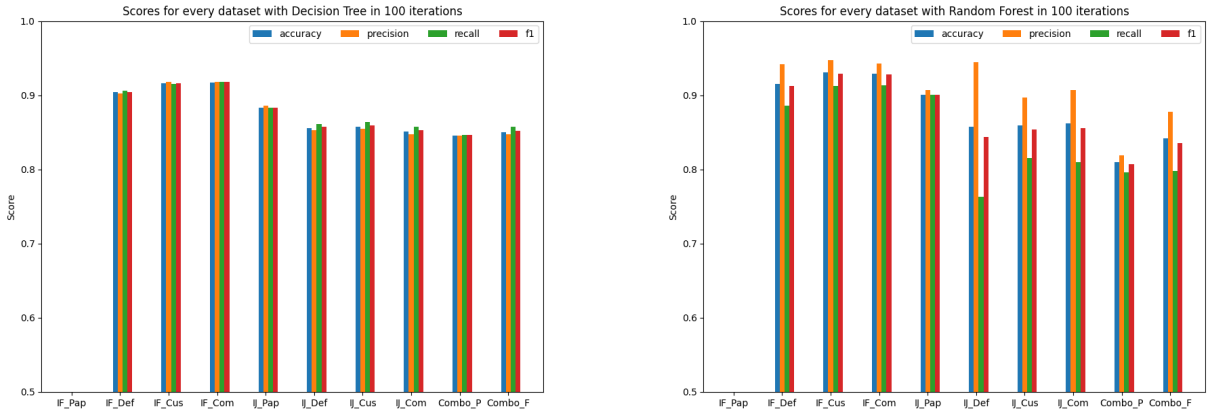
| | DT | RF | SVM | NB (B) | NB (G) | LR | MP |
|---|---|---|---|---|---|---|---|
| *InstaFake (paper)* | | | *0.82* | *0.68* | *0.98* | *0.7* | *0.84* |
| InstaFake (our default) | 0.906 | 0.886 | 0.828 | **0.929** | 0.928 | 0.866 | 0.929 |
| InstaFake (custom) | 0.915 | 0.912 | **0.902** | 0.812 | **0.93** | 0.904 | 0.934 |
| InstaFake (compatibility) | **0.918** | **0.913** | 0.883 | 0.899 | 0.895 | **0.907** | **0.938** |
| *IJECE (paper)* | *0.883* | *0.901* | | | *0.731* | *0.809* | *0.817* |
| IJECE (our default) | 0.862 | 0.763 | **0.782** | 0.542 | 0.485 | **0.796** | **0.853** |
| IJECE (custom) | **0.864** | **0.816** | 0.74 | **0.548** | 0.74 | 0.755 | 0.76 |
| IJECE (compatibility) | 0.858 | 0.81 | 0.739 | 0.449 | **0.781** | 0.776 | 0.805 |
| Combo (partial) | 0.847 | 0.796 | 0.658 | 0.119 | **0.79** | 0.733 | 0.691 |
| Combo (full) | **0.857** | **0.797** | **0.711** | **0.437** | 0.556 | **0.75** | **0.801** |

Tabella 5: Recall per tutti i metodi con i vari dataset.

| | DT | RF | SVM | NB (B) | NB (G) | LR | MP |
|---|---|---|---|---|---|---|---|
| *InstaFake (paper)* | | | *0.86* | *0.78* | *0.67* | *0.75* | *0.817* |
| InstaFake (our default) | 0.904 | 0.913 | 0.888 | 0.671 | 0.712 | 0.878 | 0.879 |
| InstaFake (custom) | 0.916 | **0.929** | **0.922** | **0.772** | **0.779** | **0.916** | **0.931** |
| InstaFake (compatibility) | **0.918** | 0.928 | 0.907 | 0.74 | 0.852 | 0.881 | 0.896 |
| *IJECE (paper)* | *0.883* | *0.901* | | | *0.721* | *0.809* | *0.817* |
| IJECE (our default) | 0.857 | 0.844 | **0.802** | **0.632** | 0.597 | **0.804** | 0.847 |
| IJECE (custom) | **0.859** | 0.854 | 0.783 | **0.632** | 0.712 | 0.79 | 0.843 |
| IJECE (compatibility) | 0.853 | **0.855** | 0.774 | 0.561 | **0.721** | 0.782 | **0.859** |
| Combo (partial) | 0.846 | 0.807 | 0.747 | 0.203 | **0.667** | 0.771 | 0.745 |
| Combo (full) | **0.852** | **0.835** | **0.76** | **0.551** | 0.597 | **0.772** | **0.86** |

Tabella 6: F1-score per tutti i metodi con i vari dataset.

indicando un chiaro successo della soluzione proposta. Per quanto riguarda le Random Forest, la configurazione custom performa meglio della configurazione di default per tutte le metriche sul dataset InstaFake. Riguardo a IJECE, si è presentato invece un calo nella precision, accompagnato da un aumento nella recall. In altre parole, il modello allenato utilizzando il dataset originale era molto bravo a non indicare come fittizi account che non lo fossero, riconoscendo tuttavia un numero inferiore di account effettivamente fittizi. Il modello allenato utilizzando la configurazione proposta, dall'altro lato, riconosce un numero maggiore di account fittizi, al costo di segnalare più spesso come account fittizi account che in realtà non lo sono. Il valore di F1-score presenta comunque un trend positivo, indicando come la distribuzione fra precision e recall sia comunque migliore per il modello custom, andando a confermare assieme al valore di accuracy il successo della soluzione proposta. In generale, i Decision Tree riportano valori più omogenei rispetto alle Random Forest, le quali hanno precision più alta a costo di una recall più bassa, indice di un rischio maggiore di falsi negativi. Risulta inoltre interessante come la configurazione compatibile di entrambi i dataset riporti risultati molto simili rispetto alla configurazione custom, soprattutto per quanto riguarda i Decision Tree, cosa che si riflette nei risultati ottenuti unendo i due dataset, dimostrando che l'utilizzo di questa combinazione di attributi accompagnata dall'implementazione di questa tecnica è in grado di generalizzare sufficientemente bene, facendo uso di un numero inferiore di attributi comuni, e fornendo comunque dei buoni risultati.




(a) Decision Tree

(b) Random Forest

Figura 8: Risultati ottenuti utilizzando i metodi di Decision Tree e Random Forest

Support Vector Machine Anche per quanto riguarda le SVM, il modello allenato utilizzando la configurazione custom si è rivelato vincente su tutte le metriche rispetto a quello di default negli esperimenti relativi al dataset InstaFake. Tuttavia, lo stesso non si può dire per il dataset IJECE. Essendo le SVM molto dipendenti dalla loro configurazione, ed in particolare dal kernel utilizzato e dal valore del parametro di regolarizzazione $C$ come descritto in sezione 2.1.2, sono stati tentati vari esperimenti di comparazione variando il valore di quest'ultimo fra 0.5 e 5.0, considerando che ad un valore inferiore di questo corrisponde una maggiore regolarizzazione, tuttavia con scarso successo. Questo è probabilmente dovuto dalla scelta di utilizzare la classe `sklearn.svm.LinearSVC`, la quale come specificato nella documentazione è meno sensibile al variare di $C$. Consigliamo quindi un'indagine utilizzando kernel differenti (ricordiamo che la scelta di limitarci a questa funzione è data dalla complessità computazionale). Come le Random Forest, anche le SVM presentano in generale una precision più alta della recall, presentando perciò il rischio di una quantità maggiore di falsi negativi. I risultati ottenuti utilizzando come metodo le SVM con $C = 1.0$ sono riportati in Figura 9.

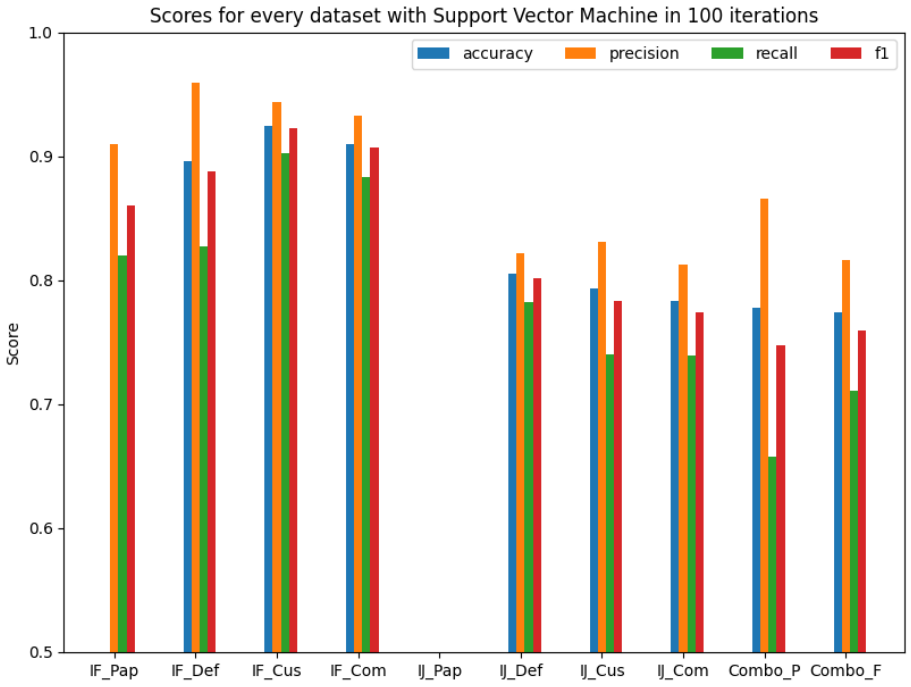
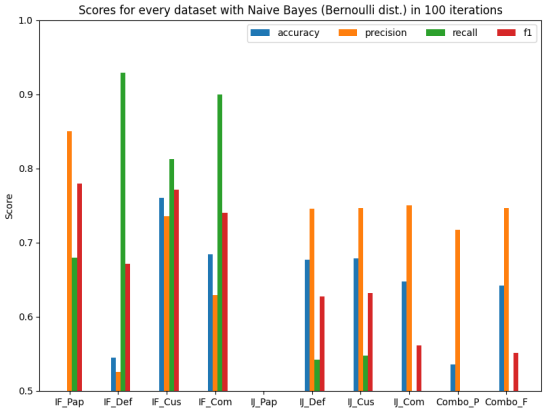


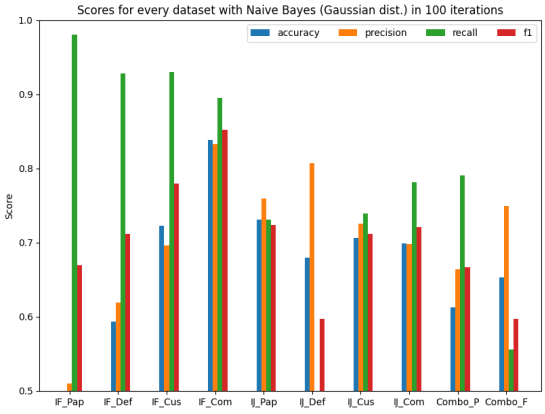

Figura 9: Risultati ottenuti utilizzando come metodo le SVM

Naive Bayes Fra i metodi proposti, Naive Bayes è sicuramente quello che ha riportato i risultati meno soddisfacenti, in linea comunque con ciò che già emerge negli articoli di partenza, dimostrando come la tecnica non sia adatta alla soluzione di questo problema. Ciò risulta evidente dalla Figura 10. Infatti, entrambe le alternative proposte riportano risultati tali da non considerare i modelli sufficientemente attendibili, in particolare relativamente al dataset IJECE per cui sia accuracy che F1-score in entrambi i metodi non hanno mai raggiunto il 75%. Al fine di trovare il modello migliore, si è sperimentato con il parametro *alpha*, descritto nella documentazione come parametro di *smoothing* additivo di Laplace/Lidstone, provando valori sia molto piccoli (fra 1.0 e $1e^{-10}$), sia molto grandi (fra 100 e 1000), mostrando come l'aumentare di questo valore portasse a un miglioramento dei risultati che si è stabilizzato a 500. Nonostante le deboli performance

di questa soluzione, per entrambi i dataset l'utilizzo della configurazione proposta porta a dei risultati migliori rispetto all'utilizzo di quelli originali, per entrambe le alternative di Naive Bayes prese in considerazione. Riguardo alla distribuzione fra precision e recall, la prima si presenta di molto più alta rispetto alla seconda usando la distribuzione di Bernoulli per quanto riguarda IJECE (Figura 10a), a differenza delle altre configurazioni in cui si presenta invece il problema opposto, ovvero una recall quasi sempre di molto più alta della precision (in linea con [2], in cui la recall è quasi doppia rispetto alla precision), indicando una quantità molto elevata di falsi positivi (ricordiamo che per ottenere un valore di recall del 100% sarebbe sufficiente, al modello, indicare tutti i campioni come positivi; è l'esempio di *IF_Def*, in cui la recall è vicina al 95% e tuttavia la precision è di poco superiore al 50%). Infine, risulta interessante sottolineare come i valori migliori siano stati ottenuti con *IF_Com*, indicando come nel caso dei Decision Tree come un ulteriore raffinamento sugli attributi del dataset InstaFake sia probabilmente possibile.




(a) Naive Bayes: Bernoulli Distribution




(b) Naive Bayes: Gaussian Distribution

Figura 10: Risultati ottenuti utilizzando come metodo Naive Bayes

Logistic Regression Figura 11 mostra i risultati ottenuti con il metodo di Logistic Regression. Risulta interessante a colpo d'occhio come, per quanto riguarda InstaFake, con la configurazione proposta i valori siano di molto migliori in confronto a quelli riportati nel relativo articolo, sia utilizzando il dataset originale che utilizzando la combinazione di attributi proposta. Quest'ultima, ancora una volta, si rivela vincente nei confronti della prima per InstaFake, mentre riguardo a IJECE si verifica un piccolo calo nelle performance, seppur minimo. Riguardo ai parametri del classificatore, durante la fase di comparazione sono state provate varie configurazioni, ad esempio variando il parametro di regolarizzazione $C$, tuttavia con scarso successo. Abbiamo deciso di selezionare un numero massimo di iterazioni molto alto (5000) in modo da far convergere in ogni caso preso in considerazione i modelli. Tuttavia, ciò concorre a rendere il tempo di allenamento del modello molto lungo, il più alto fra quelli utilizzati, e portando quindi a preferire in molti casi l'utilizzo di altri modelli.

Multilayer Perceptron Infine, Figura 12 riporta i risultati ottenuti con il Multilayer Perceptron utilizzando due strati di 8 neuroni, il quale si conferma una buona strategia sia per InstaFake che per IJECE, seppur quest'ultimo sembra presentare una precision abbastanza più alta della recall, indicando quindi una preferenza nel non indicare falsi

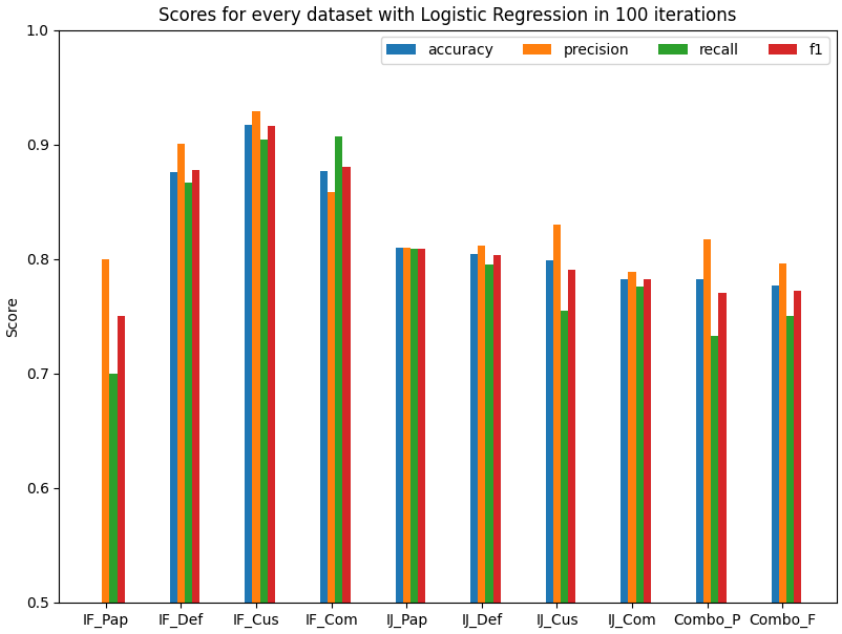



Figura 11: Risultati ottenuti utilizzando come metodo Logistic Regression

positivi a costo di non riconoscere utenti fittizi. La configurazione degli attributi proposta vede un chiaro vantaggio rispetto a quella origninale per quanto riguarda InstaFake, mentre per quanto riguarda IJECE seppur questa presenti performance simili per le due configurazioni, con *IJ_Cus* che guadagna leggermente in accuracy a costo dell'F1-score, questo vantaggio viene recuperato con la versione compatibile, indicando comunque un miglioramento rispetto all'articolo.

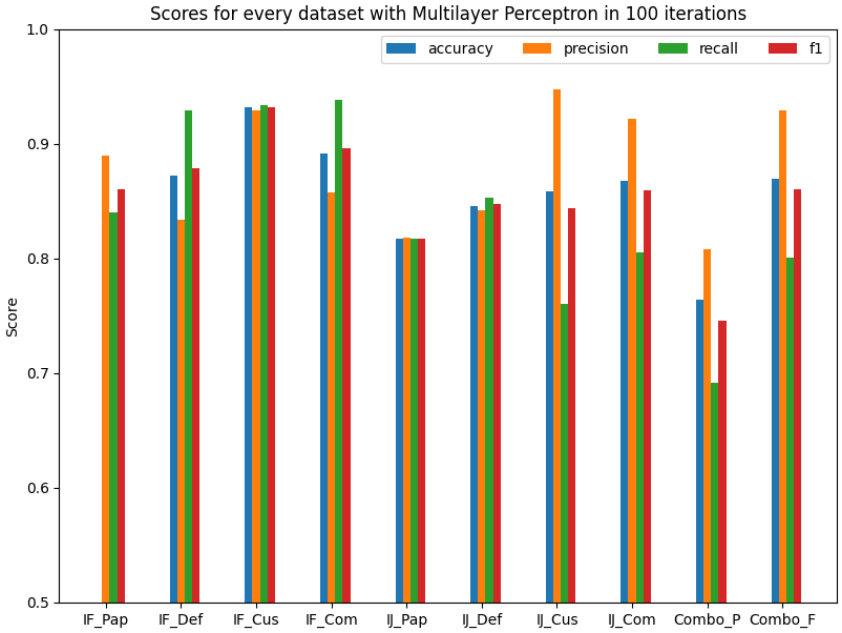



Figura 12: Risultati ottenuti utilizzando come metodo Multilayer Perceptron

# 4 Discussione e conclusioni

## 4.1 Risultati ottenuti

Lo scopo di questo progetto era quello di verificare se fosse possibile ottenere risultati migliori rispetto a quanto riportato in letteratura al problema della rilevazione di account fasulli sulla piattaforma social Instagram, utilizzando dataset già esistenti. Se si prendono in considerazione i dati della Tabella 1, si può osservare come - in media tra tutti i metodi analizzati - i dataset custom ottengano risultati migliori rispetto alle loro controparti default. InstaFake custom, in particolare, supera nettamente i risultati default, e lo stesso vale per IJECE, seppur marginalmente: bisogna però considerare come i risultati su quest'ultimo vengano ottenuti rimuovendo attributi considerati utili dall'articolo [3], indicando come la loro rimozione abbia portato beneficio ed essi fossero perlomeno poco rilevanti. Per quanto riguarda i dataset combinati, il dataset totale mostra risultati migliori rispetto al parziale. Il bilancio del progetto è positivo, essendo riusciti a migliorare quanto già esisteva rimuovendo attributi dove necessario e aggiungendole dove richiesto. Da segnalare inoltre come all'interno di [3] non siano contenute informazioni fondamentali per replicare e verificare sperimentalmente i risultati presentati, e per questo motivo i valori di riferimento dell'articolo vengono riportati solo a scopo informativo, non essendo stati in grado di replicarli fedelmente. Il nostro studio ha mostrato come i metodi migliori di classificazione per questo problema siano i Decision Tree e le Random Forest, rimanendo in linea con quanto visto all'interno dell'articolo [3].

## 4.2 Limitazioni

Il progetto presentato è basato sui dataset dei paper [3] e [2], di conseguenza bias presenti nei dati sono stati *ereditati* dal nostro progetto. Analisi statistiche riportate all'interno di questo documento (sottosottosezione 2.1.1) mostrano, inoltre, una significativa differenza nei contenuti dei due dataset, dovuta alle diverse strategie utilizzate per la raccolta dati e un possibile errore della componente umana che si è occupata di classificarli. Queste differenze sono rese evidenti nei dataset *Combo*: sebbene i modelli relativi ai singoli dataset in configurazione compatibilità forniscano metriche elevate, questo trend va ad interrompersi nel momento in cui vengono combinati, ottenendo risultati misti. Il progetto è sufficientemente leggero in termini di risorse richieste da poter essere eseguito su computers non particolarmente performanti, e la velocità con cui un verdetto può venire emesso è legata all'algoritmo che si sceglie di usare: l'approccio Multilayer Perceptron è sicuramente il più rapido, in quanto la rete viene già fornita pre-addestrata, mentre altri approcci richiedono un tempo di addestramento iniziale.

## 4.3 Lavori futuri

Al fine di espandere il progetto con l'intenzione di migliorarne l'efficacia nel riconoscimento di account falsi e nel renderlo più facilmente fruibile per un qualsiasi utente della rete, sono proposti i seguenti lavori futuri:

- **Creazione di un dataset ex-novo** Il nostro lavoro è stato limitato dalla quantità e varietà di caratteristiche presenti nei dataset che abbiamo recuperato, e sarebbe interessante vedere cosa si possa riuscire a fare con differenti attributi a disposizione, raccogliendo dati direttamente usando le API di Instagram.

- **Miglioramento della fruibilità** Allo stato attuale, il nostro progetto non è facilmente utilizzabile da un utente qualsiasi della rete, e comunque non rappresenta al momento uno strumento pratico per il riconoscimento di account fasulli. Uno sviluppo interessante consisterebbe nel creare un'applicazione facilmente raggiungibile da un utente che, dato un link al profilo di un utente, lo analizzi tramite le API di Instagram ed emetta un verdetto.

# Riferimenti bibliografici


[1] Koosha Zarei, Reza Farahbakhsh, and Noël Crespi. How impersonators exploit instagram to generate fake engagement? In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[2] Fatih Cagatay Akyon and M Esat Kalfaoglu. Instagram fake and automated account detection. In *2019 Innovations in intelligent systems and applications conference (ASYU)*, pages 1–7. IEEE, 2019.

[3] Kristo Purba, David Asirvatham, and R.K. Murugesan. Classification of instagram fake users using supervised machine learning algorithms. *International Journal of Electrical and Computer Engineering (IJECE)*, 10:2763, 06 2020.

[4] Jinus Bordbar, Mohammadreza Mohammadrezaie, Saman Ardalan, and Mohammad Ebrahim Shiri. Detecting fake accounts through Generative Adversarial Network in online social media. *arXiv e-prints*, page arXiv:2210.15657, October 2022.

[5] Manojit Chakraborty, Shubham Das, and Radhika Mamidi. Detection of Fake Users in SMPs Using NLP and Graph Embeddings. *arXiv e-prints*, page arXiv:2104.13094, April 2021.

[6] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984.

[7] Roger J Lewis. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer, 2000.

[8] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[9] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[10] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 08 2008.

[11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[12] Jan Salomon Cramer. The origins of logistic regression. 2002.

[13] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, dec 1997.

[14] C.M. Bishop. *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics. Clarendon Press, 1995.